Homework #4 (A.K.A. CS520 Final Exam 2014)

CS520 / KAIST Fall 2015

Due: December 9, 2015 2:30PM

1. In λ-Calculus with normal-order reduction rules, encode CONS, CAR, and CDR operators (as in Scheme or Lisp). The operators are such that

   CAR(CONS e_1 e_2) = e_1
   CDR(CONS e_1 e_2) = e_2

   Verify your encodings of CAR and CONS by showing an example.

2. As mentioned in lecture notes, in eager evaluation we cannot encode the if-branch and recursion the same way as in normal-order evaluation. How should we encode the if-branch and recursion in eager evaluation? Complete the encoding definitions:

   $\underline{\text{if}\ \ e_1\ \ e_2\ \ e_3} \equiv$ <???>
   $\underline{\text{fac}} \equiv$ <???>

3. Consider the following applicative (eager-evaluation) language.

   $$\begin{array}{lll} \boldsymbol{e} & ::= & \boldsymbol{n} \quad | \quad \boldsymbol{x} \\ & | & \texttt{if0}\ \boldsymbol{e}\ \boldsymbol{e}\ \boldsymbol{e} \\ & | & (\boldsymbol{e},\ \boldsymbol{e}) \quad | \quad \boldsymbol{e}.1 \quad | \quad \boldsymbol{e}.2 \end{array}$$

   Define the cps transformation function $[\bullet] \in \texttt{Expr}\rightarrow\texttt{Expr}$ for (e_1, e_2) and $e.1$.

   Ex) $[\texttt{if0}\ e_1\ e_2\ e_3] = \lambda\kappa.\,[e_1](\lambda v.\ \texttt{if0}\ v\ [e_2]\kappa\ [e_3]\kappa)$

4. In class note “class8”, we define a simple language and derive a simple type system. We are going to extend the language to include integer addition, if zero, and recursive functions, whose semantics is as usual (Refer to class note “class6”.)

   $$\begin{array}{l} \boldsymbol{e} ::= \boldsymbol{x} \ |\ () \ |\ \lambda \boldsymbol{x}.\boldsymbol{e} \ |\ \boldsymbol{e}\ \boldsymbol{e} \ |\ \texttt{let}\ \boldsymbol{x} = \boldsymbol{e}\ \texttt{in}\ \boldsymbol{e} \\ \quad |\ \boldsymbol{e} + \boldsymbol{e} \ |\ \texttt{rec}\ \boldsymbol{f}\ \lambda \boldsymbol{x}.\boldsymbol{e} \ |\ \texttt{if0}\ \boldsymbol{e}\ \boldsymbol{e}\ \boldsymbol{e} \end{array}$$

Accordingly, we want to extend our simple type system. The type inference rule for "***e + e***" is

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}$$

The semantics of " **if0 *e1 e2 e3*** " is "if e1 is zero, then e2 else e3" as usual. The type inference rule is

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \texttt{if0}\ e_1\ e_2\ e_3 : \tau}$$

a. Based on the rule defined above, define the constraint generation function

$V(\Gamma,\ \texttt{if0}\ e_1\ e_2\ e_3,\ \tau)$

b. We are going to prove the type of the following program with provability of unification logic. <u>What is τ of the following expression?</u> Generate a system of constraint equations with functions V. Solve the equations.

$$\phi \vdash \lambda z.\texttt{let}\ f = \lambda x.(\texttt{if0}\ x\ 1\ (z\ x))\ \texttt{in}\ (f\ 1) : \tau$$

c. <u>What is τ</u> for the following expression? Then prove the following judgment, that is, build a proof tree.

$$\phi \vdash \lambda y.((\lambda f.(f\ 4))(\lambda x.y)) : \tau$$

5. We learned **let**-polymorphic type system in class note c**lass9**. <u>First, write down what the type τ is</u> in the following expression? <u>Second, prove it</u>, that is, show the proof tree. The type inference rule for **(e, e)** is

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}.$$

$$\phi \vdash \lambda x.\ \texttt{let}\ t = \lambda x.\lambda y.(x,y)\ \texttt{in}\ ((t\ 1\ x),\ (t\ x\ 2)) : \tau$$